Open access • Proceedings Article • DOI:10.1109/SMARTWORLD.2018.00108

A Domain-Specific Approach to Unifying the Many Dimensions of Context-Aware Home Service Development — Source link

Nic Volanschi, Adrien Carteron, Charles Consel

Published on: 08 Oct 2018 - Ubiquitous Intelligence and Computing

Topics: Service design, Software architecture, Database-centric architecture and Context (language use)

A Domain-Specific Approach To Unifying The Many Dimensions of Context-Aware Home Service Development

Nic Volanschi
Inria, Bordeaux, France
eugene.volanschi@inria.fr

Adrien Carteron
Inria, Bordeaux, France
adrien.carteron@aquilenet.fr

Charles Consel
Bordeaux INP & Inria, Bordeaux, France
charles.consel@inria.fr

***Abstract*—Developing context-aware homes involves a range of stakeholders, addressing many dimensions such as service design and development, infrastructure deployment, and maintenance. Such a variety of dimensions often translate into heterogeneous, low-level, silo-based processing of sensor data to extract context information.**

This paper analyzes a range of existing data processing layers in the domain of aging in place to identify key concepts and operations specific to context-aware processing. Based on this analysis, we introduce a context-aware, domain-specific language and its software architecture, which allow to put in synergy the stakeholders of a context-aware home by providing them with a unified approach to designing and developing services. Our approach offers context aware-specific abstractions and notations, within a data-centric and data-driven paradigm.

We have validated our approach by applying it to an assisted living platform for aging in place, deployed in the home of 129 users. In particular, we used our domain-specific language to re-implement 53 existing services, originating from the stakeholders of the assisted living platform. These services were deployed and successfully tested for their effectiveness in performing the specific tasks of the stakeholders, such as detection of daily activities, user risk situations, or sensor failures.

***Index Terms*—context-aware, domain-specific, data-centric, data-driven, aging in place, programming models**

I. Introduction

The notion of *context* is fundamental to the field of pervasive computing and encompasses a range of dimensions, including the physical world, an individual, their activities, and their technologies [1]. A major focus of the research on context awareness has been the *home* (*e.g.,* [2], [3]). This scope involves many context dimensions, including physical interactions with the environment (*i.e.,* sensors), digital interactions with the user environment (*e.g.,* email, calendar), status of the many components of the pervasive computing infrastructure (hardware, software, and network), application-specific concerns (*e.g.,* activity detection). When it comes to context-aware homes for the general population, a recurring challenge is to identify what services users would need [4], and more generally, to develop methods to gather and analyze these needs [5].

However, when focusing on older adults, this population is ready to benefit from context-aware homes to support aging in place. A pervasive computing environment has the potential to deliver assistive services 24/7 and address the needs of older adults, whose nature can be of utmost importance to ensure independent living (*e.g.,* [6]). These services mainly consist of 1) detecting and reminding daily activities (*e.g.,* meal preparation, self-care, going to bed) to maintain the user's functional status [7] and 2) monitoring potentially hazardous situations (*e.g.,* cooker, entrance door) to make the user safe [6]. Because it crosscuts various fields, a context-aware home dedicated to aging in place involves a variety of stakeholders to design and develop assistive services, as well as to deploy and maintain the underlying infrastructure. Stakeholders include older users, caregivers, aging experts, health professionals, application developers, and maintenance technicians. This considerable diversity of stakeholders raises correspondingly diverse context dimensions. Typically, each stakeholder develops their own, silo-based approach to extracting their specific context information from sensed data. This approach prevents any synergy between stakeholders such as code/experience sharing, and favors the duplication of solutions for detecting similar event patterns.

We analyzed existing data processing layers of multiple stakeholders, extracted from an assisted living platform, deployed in 129 single-occupant homes of older adults, aged 82 years old on average [8]. This case study thus consists of a range of services, whose usefulness has been validated in practice on a daily basis. Our analysis identifies commonalities and variabilities of these layers, revealing key concepts and operations specific to context-aware processing.

Our Approach: Based on this analysis, we introduce a context-aware, domain-specific language and its software architecture, which provide a conceptual framework and a tool to unify the design and the development of home services. To unify heterogeneous sources of sensed data, ranging from hardware devices to software components, our approach promotes a processing paradigm, which is data centric and data driven. Specifically, our approach is *data centric* to provide a canonical view of sensed data to a range of services, spanning maintenance services consuming low-level device status, to caregiving-specific services using high-level activity measures. Our approach is *data driven* in that services are defined in terms of rules processing events and states.

To unify the way services are developed, our approach introduces abstractions and notations that are specific to context-

aware processing. The resulting *domain-specific language* (DSL) covers the needs of the stakeholders and provides an abstraction layer over underlying, well-established concepts and technologies, such as Allen's algebra to express sequences of interactions [9] and complex-event-processing engines to efficiently process the rules generated from DSL services (*e.g.,* [10]). Furthermore, we envision that our language can serve as a high-level stepping stone to introduce end-user programming languages for stakeholders with no computer-programming background.

To validate our approach, we have re-implemented 53 services ranging over all the stakeholders of the assisted living platform under study. These new services were deployed and successfully tested for their effectiveness in performing the specific tasks of the stakeholders: detection of daily activities, user risks, and sensor failures.

To summarize, this paper makes the following contributions.

Domain analysis. We provide an analysis of context-aware processing layers in the domain of aging in place. From this analysis, we identify key concepts and operations specific to context-aware processing.

Domain-specific language. We introduce a language, specific to developing context-aware services, providing high-level abstractions and notations. Underlying this language is a data-centric and data-driven paradigm that allows services from a range of stakeholders to uniformly process heterogeneous sources of sensed data.

Compiler. We have implemented a compiler for our DSL that maps high-level rules into low-level requests, crunched by an event-processing engine.

Validation. We applied our approach to re-implement 53 services, ranging over all the stakeholders of an assisted living platform dedicated to aging in place. The resulting services are expressed at a high level and are effective in performing the tasks of the stakeholders.

II. Domain analysis

Context-aware homes for aging in place are still in their infancy and the path to adoption is being actively researched [11]. The literature include few articles reporting on the deployment of assisted living platforms in the wild (*e.g.,* [12]). Fortunately, we have been able to leverage the *HomeAssist* project to conduct a domain analysis of context-aware homes for aging in place.

A. HomeAssist: A Context-Aware Home for Aging in Place

HomeAssist is an assisted living platform that provides a catalog of assistive applications, supporting and monitoring daily activities, safety and social participation[1] [8]. HomeAssist has been deployed in 129 single-occupant homes of community-dwelling older adults, 82 years old on average. The duration of this field study is 12 months.

HomeAssist is a perfect case study on which to build a unifying approach to developing context-aware home services dedicated to aging in place. It matches all the requirements to pursue our goal: 1) it is deployed in the wild in real homes; 2) it supports aging in place for frail users with pressing needs; 3) the field study is long enough that maintenance and evolution problems must be properly handled; 4) it is deployed at a large enough scale that administering context-aware homes need to be supported by services; 5) existing services reflect a range of needs expressed by stakeholders, spanning older users, caregivers, occupational therapists, psychologists, human factors experts, installation and maintenance technicians, and computer scientists.

Let us further describe this platform to delimit the scope of the issues raised by the context-aware services. HomeAssist consists of a client-server architecture, where the server runs as many virtual machines as they are context-aware homes. Each virtual machine executes the assistive services selected by the user and their caregiver. These assistive services are fed with sensing data sent via Internet by a gateway, deployed in the context-aware home. This gateway gathers information from the sensors placed at strategic locations in the older adult's home to monitor their daily activities (see Caroux *et al.* for more details [7]). As well, the gateway channels actions from the services to the home's actuators. In the HomeAssist field study, a typical home consists of 4 contact sensors (entrance door, fridge, drawers, cabinets), 6 motion detectors (entrance area, kitchen, bathroom, *etc.*), and 2 smart plugs, which measure the electricity consumption and turn on/off a connected appliance (microwave, light path, coffeemaker, *etc.*). The number and type of devices can vary depending on the configuration of the home and the activities to be monitored. Finally, the home is equipped with a stationary tablet, placed at a central location in the home and always connected to a power outlet. This tablet is dedicated to interacting with the user via notifications emitted by assistive applications, which need to alert the user of a given situation (*e.g.* unattended entrance door left open) [13].

B. Scenarios To Support Aging In Place

We now present four scenarios that illustrate the spectrum of stakeholders and concerns involved in supporting aging in place with a context-aware home (see Figure 1). The first scenario addresses the safety concern of the older adult. It consists of monitoring the entrance door to ensure that it is not left open for too long without being attended. The second scenario relates to a caregiver's need to monitor a user's daily activities, and in particular, their eating routines. The last two scenarios address the home technician's concerns to keep the context-aware home in an operational state. The first maintenance scenario detects inconsistent values produced, or values omitted, by the motion detector of the kitchen. This situation is discovered by cross-checking the motion detector information with that of the contact sensor of the

[1]HomeAsssist applications were designed to be proactive. They aim at prolonging aging in place by preventing decline (e.g., by ensuring that daily routines are regularly performed) and detecting early causes of decline (e.g., social isolation). Thus, they are complementary to more classical applications such as fall detection, which aim at detecting the *consequences* of decline.

Stakeholder	Domain	Name	Description
Older adult	Safety	Door Alert	Entrance door is open and is unattended for 5 minutes
Caregiver	Daily Activities	Reheating A Frozen Meal	Freezer gets used and stove gets turned on within 10 minutes or Freezer gets used during stove is on, during lunch time (or dinner time)
Home Technician	Maintenance	Presence Dependency	Whenever the cupboard gets opened in the kitchen, a presence in the kitchen is true
Home Technician	Maintenance	Communication Failure	A sensor fails to communicate for 24 hours and its status does not get updated

Fig. 1. Example scenarios for assistive services

cupboard (or any other sensor located in that room). The second maintenance scenario detects whether a sensor fails to communicate; this situation occurs when the sensor has a drained battery or is malfunctioning.

These scenarios offer a glimpse at the kind of context-aware services needed to support aging in place. Some services, such as "Door Alert", can fit most older users. Other services may target daily activities that require a level of personalization for an effective monitoring. This is illustrated by the meal preparation activity and the "Reheating a Frozen Meal" scenario.

Similarly, for maintenance scenarios, "Communication Failure" applies to any context-aware home, whereas "Presence Dependency" requires to instantiate the consistency rules with respect to the locations of sensors.

C. Commonality and Variability Analysis

To analyze stakeholder needs, we have examined a range of services offered by HomeAssist to identify their commonalities and variabilities. These services were developed in Java using a tool-based design methodology [14], [15].

We now present the outcomes of this analysis that take the form of high-level, domain-specific concepts. These concepts will pave the way to our domain-specific approach presented in the next section.

Commonalities. All services refer to a notion of *environment* from which to perform measures. These measures encompass interactions in both a physical environment (*e.g.,* a motion detected) and a digital one (*e.g.,* an event reminder issued by a calendar). Additionally, we identified two complementary concepts related to an environment: events and states. On the one hand, an *event* defines an environment measure that changes (*e.g.,* door *gets* closed/open) – events are underlined with a dashed line in the scenarios of Figure 1. On the other hand, a *state* makes an event persistent across time (*e.g.,* door *is* open) – states are underlined with a solid line in Figure 1. Once these unitary concepts were identified, we found specific ways in which they can be combined, revealing *composition* commonalities. Specifically, the combination of environment measures can define an *order* in which interactions must occur and their *duration* – these constraints are underlined with a dotted line in Figure 1.

Let us further study these commonalities by examining their range of variability.

Variabilities. Environmental measures may be realized by a variety of entities, including hardware (*e.g.,* sensor), software (*e.g.,* calendar), local (*e.g.,* door), and remote (*e.g.,* new email messages). The abstraction level of the environmental measures varies widely. For example, an event may be produced by a sensor, as soon as a motion is detected in a room. Alternatively, a sensor may detect the state of a room being occupied, excluding room-to-room transfers.

Regarding composition, several order constraints were observed between interactions, including an interaction *preceding* another one, an interaction occurring *during* another one, and an interaction *overlapping* another one. Not all of these constraints are applicable to any kind of interactions (*i.e.,* event and state). For example, only two states may overlap, whereas two events cannot. Indeed, in practical scenarios, events are viewed as occurring sequentially, not simultaneously.

III. A Domain-Specific Approach

We now present the main stages of our domain-specific approach to developing context-aware services dedicated to aging in place. This approach is depicted in Figure 2.

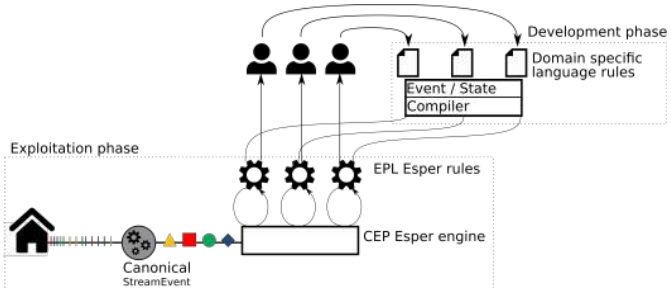

Fig. 2. Overall view of our domain-specific approach

1) Service definition: The first step is initiated with the stakeholders that express scenarios of services, as illustrated earlier. These scenarios are directly written in our domain-specific language (see next section) by the stakeholders, if they have the proper background, or by a service developer.

2) Service compilation: The high-level service is compiled into low-level rules written in an event processing language. These rules make explicit the domain-specific concepts, such as states that are compiled into a combination of events and related operations.

3) Service execution: When deployed, the rules are added to a Complex Event-Processing (CEP) engine. Our rule execution engine is based on Esper, an open source CEP developed by EsperTech.[2] It offers Java and C# interfaces to develop event-based programs. We chose Esper because it is a popular CEP engine, used both in industry and research. Esper contains a declarative domain-specific language for CEP, called EPL (for Event Processing Language). EPL allows to describe patterns of events to be recognized in an online stream of events, using operators for ordering events, time constraints, alternatives, *etc.* Esper does not offer a concept of state; it only handles events and requires extra machinery to manage a state. In our implementation, we run the Esper engine with respect to EPL rules, compiled from our DSL rules, and a stream of events from the context-aware home, formatted in a canonical form.

Canonical form: The canonical form of sensed data produced by context-aware homes allows to process them uniformly, disregarding their original heterogeneous formats. In our implementation, the canonical form of data, called *StreamEvent*, is introduced as an abstraction layer above the flow of sensed data events. In this representation, each event consists in a 4-tuple consisting of the kind of event, its location, its value, and the timestamp of its occurrence.

IV. A DSL For Context-Aware Services

In this section, we introduce our DSL for developing context-aware services, named Maloya. This DSL is dedicated to describing contexts in terms of states and events, and operators to combine them. Figure 3 presents the syntax of Maloya, as well as its informal semantics. Each construct of the language is presented on the left-hand side; its corresponding graphical representation is displayed on the right-hand side. This graphical representation visualizes events and states, and how operators combine them. A state is represented as a rectangle-shaped signal, lasting between its starting and ending points. An event is represented as a spike signal, with merged starting and ending points. Underneath each DSL construct is its translation into a core DSL, which can be viewed as an abstract syntax.

A. Events and States

In our DSL, testing an event or a state is expressed as follows: for an event, `p becomes v` ; for a state, `p is v`. Where, in both cases, `p` is a sensor name (hardware or software) and `v` is a value in the range of the sensor. The event `p becomes v` occurs when the sensor `p` signals a value `v`, if its previous value was different. The state `p is v` starts precisely when the event `p becomes v` occurs; it ends when the event `p becomes v'` occurs, where $v' \neq v$. We view the period during which a state holds as the *time interval* of a state. This notion is generalized for events by viewing them as defining a zero-length time interval. The notion of time interval is used to define operators and their semantics.

[2]http://www.espertech.com/esper/

A *rule* in our DSL consists of an operator applied to states and/or events. All operators return events. More precisely, an operator yields a success event when the context described by the application of the operator is detected. Because operators return events, operators taking an event as a given argument can call a nested operator instead. On the other hand, operators taking a state as a given argument cannot call a nested operator instead. Thus, the nesting of operators is not arbitrary; it follows the results of our domain analysis. Operators are further discussed next.

B. Operators

Our operators, listed in Figure 3, are based on the operators in Allen's time interval algebra [9], viewing states and events as time intervals, as explained earlier. Specifically, Allen's operators model all possible relations between two time intervals, such as *preceding*, *during*, or *overlapping*. However, in our domain, since a context-aware home produces in principles an infinite stream of events, it may contain several occurrences of the same event. For example, an event such as lunch activity may occur many times in the stream of events produced by a home; typically, every day. Thus, a rule checking whether the lunch activity is performed during lunch time is tested repeatedly: for each occurrence of the lunch time slot. To account for this situation, we generalized Allen's operators between two intervals to account for their multiple occurrences. Allen's operators take non-empty time intervals; we generalized them to accept events, when appropriate.

Let us now review in detail the operators used in the examples of this paper. In the rule e_1 **precedes** e_2, the operator yields success every time the occurrence of event e_1 *immediately* precedes the occurrence of event e_2. This means that there must be no other occurrence of either e_1 or e_2 in between. To cover existing scenarios, we need to expand the expressiveness of this operator (and others) with optional time constraints. More precisely, we introduce two variants of **precedes**: e_1 **precedes** e_2 **within/by** `t`. The time constraint is defined by the parameter `t`. These variants specify an upper/lower bound on the time between the occurrences of its event operands.

The rule `e` **during** `s` succeeds every time event `e` occurs during state `s`. There are no time-constrained versions of this operator.

The rule s_1 **overlapping** s_2 succeeds every time state s_1 overlaps with state s_2. This means that state s_1 starts before the beginning of state s_2, and ends during s_2, as shown in the Figure 3. The time-constrained versions define an upper/lower bound on the overlapping time of the occurrence of these states.

The rule `e` **occurs while** `s` is similar to the rule `e` **during** `s`, but succeeds only for the first occurrence of event `e` during state `s`. A variant of the previous rule is s_1 **occurs while** s_2. In this case, the rule succeeds the first time state s_1 superposes at least partially with s_2. The time-constrained versions put an upper/lower time bound on the superposition of the states.

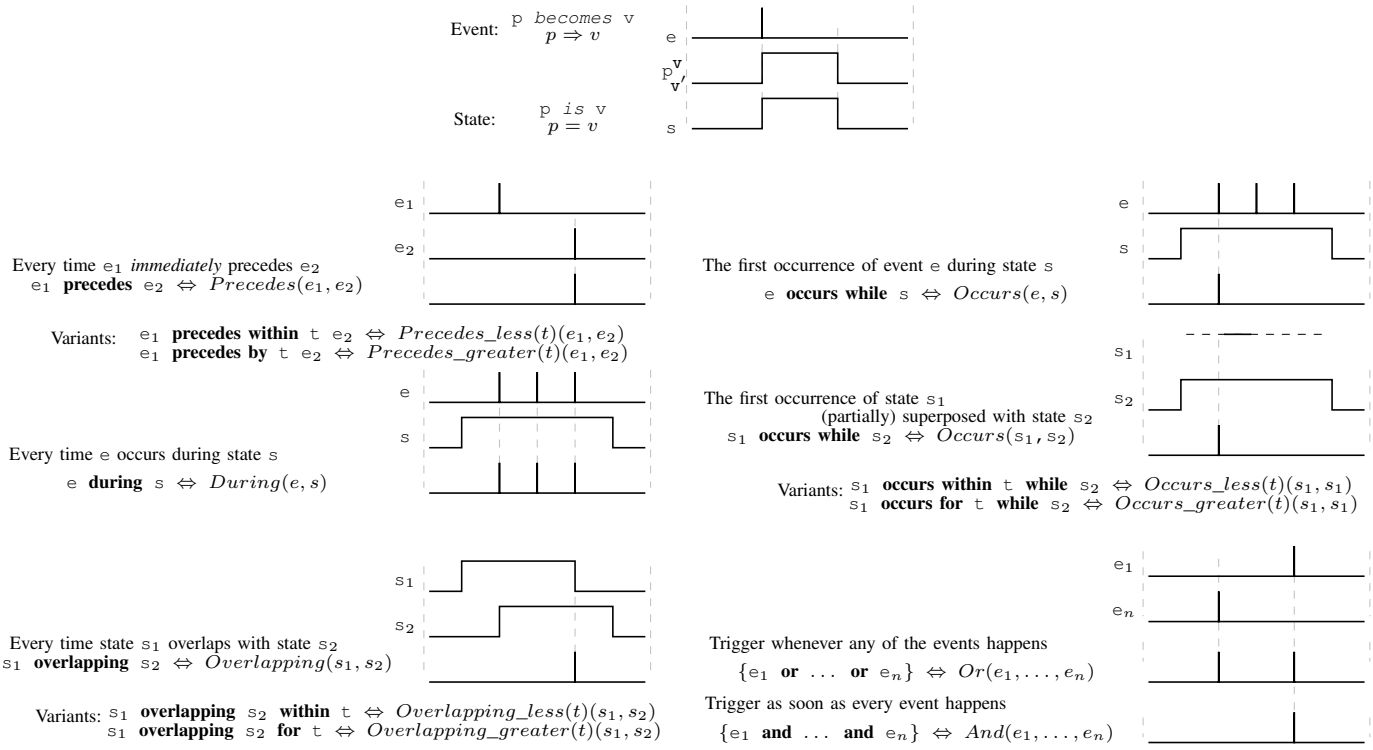

Fig. 3. DSL syntax and informal semantics

Even though Allen's operators express a range of situations, they do not cover all the needs revealed by our domain analysis; more operators are required. In particular, a disjunction of events is needed to enable alternative contexts to be expressed. A disjunctive rule is of the form $\{\texttt{e}_1$ **or** ... **or** $\texttt{e}_n\}$; it succeeds whenever any $\texttt{e}_i$ occurs. Dually, we introduced a conjunction rule of the form $\{\texttt{e}_1$ **and** ... **and** $\texttt{e}_n\}$. This rule succeeds when every $\texttt{e}_i$ occurred.

Let us now illustrate the use of our DSL operators by writing the rule for the "Lunch Reheat" activity, described earlier (Section II-B).

```
{ ( Freezer becomes open precedes
    within 10 minutes Stove becomes on )
  or
  ( Freezer becomes open occurs while Stove is on )
} occurs while LunchTime
```

Note how this specification concisely encodes two scenario variants: (1) taking a meal from the freezer, and then turning on the stove; (2) taking a meal from the freezer to put it in the stove, which is already running.

C. Compilation

Compilation is done in three main steps. The first step translates the text of DSL rules into the core DSL; this translation is defined as a one-to-one correspondence. For instance, the core DSL form of the operator e **during** s is $During(e, s)$. The translation of our running example into core DSL is as follows.

$$
\begin{aligned}
&Occurs(Or(\\
&\quad Precedes\_less(10min)(freezer => open, stove => on),\\
&\quad Occurs(freezer => open, stove = on)),\\
&\quad lunchTime)
\end{aligned}
$$

Here "=>" denotes an event that occurs and "=" denotes a state that holds.

The next compilation step consists of generating EPL pseudo-code. This pseudo-code uses only valid EPL operators, but does not yet instantiate the attributes of each event; this is done subsequently. This step involves several transformations. Firstly, as EPL does not support the notion of state, each state in a rule is translated into the sequence of corresponding events that mark the beginning and the end of the state, ordered by standard EPL operators. For instance, the state of the stove being on is translated in an EPL sequence of the stove being turned on, followed by any event of interest but not the stove being turned off. Hence, the operator "*Occurs(..., stove = on)*" is translated in EPL as

stove $\Rightarrow$ *on* $\rightarrow$ ... *and not (stove* $\Rightarrow$ *off)*

using the EPL operators "*and*", "*or*" and "$\rightarrow$", which means *followed by*. Also in this compilation phase, time constraints in the rules are translated by explicit uses of the "*timer:within*" construct in EPL for enforcing upper time bounds, and explicit uses of "*timer:interval*" EPL construct for enforcing lower time bounds. The result of this phase is the following EPL pseudo-code:

$lunchTime \Rightarrow begin \rightarrow$
$((freezer \Rightarrow open \rightarrow stove \Rightarrow on \ and$
$not\ (freezer \Rightarrow open)\ where\ timer : within(10min))$
or
$(stove \Rightarrow on \rightarrow (freezer \Rightarrow open)\ and\ not\ (stove \Rightarrow off)))$
$and\ not\ (lunchTime \Rightarrow end)$

The final step is to obtain the EPL Esper form from the EPL pseudo-code, by completely instantiating the event attributes as necessary in the stream of canonical events (*i.e.,* in StreamEvent form). To do so, we use a static table giving the attributes of each sensor in a given home:

```
"freezer":{
        "location": "Kitchen",
        "kind": "Freezer",
        "values": ["open", "close"]
}
```

Moreover, this step binds all events in an EPL formula as originating from the same home (as can be seen in the EPL constraint "`user=`X`.user`" below). Also, this step introduces "*every*" and "*every-distinct*" EPL constructs to deal with multiple occurrences of an event. As a result of these transformations, we obtain the final EPL Esper rule that is executed by the Esper engine:

```
select Cal_L_b,Fre_K_o,Sto_K_o from pattern [
  every Cal_L_b=StreamEvent(role.location='Lunch',
                            role.type='Calendar',
                            status!='end') ->
    ((every-distinct(timestamp)
      Fre_K_o=StreamEvent(role.location='Kitchen',
                          role.type='Freezer',
                          status='open',
                          user=Cal_L_b.user) ->
        Sto_K_o=StreamEvent(role.location='Kitchen',
                            role.type='Stove',
                            status='on',
                            user=Cal_L_b.user)
        where timer:within(10min)
        and not (StreamEvent(role.location='Kitchen',
                             role.type='Freezer',
                             status='open',
                             user=Cal_L_b.user)))
   or(every-distinct(timestamp)
      Sto_K_o=StreamEvent(role.location='Kitchen',
                          role.type='Stove',
                          status='on',
                          user=Cal_L_b.user) ->
      (Fre_K_o=StreamEvent(role.location='Kitchen',
                           role.type='Freezer',
                           status='open',
                            user=Cal_L_b.user))
       and not (StreamEvent(role.location='Kitchen',
                            role.type='Stove',
                            status='off',
                            user=Cal_L_b.user)))
    ) and not (StreamEvent(role.location='Lunch',
                           role.type='Calendar',
                           status='end',
                           user=Cal_L_b.user)) ]
```

Note that, even though these transformation steps may seem straightforward, there are several subtleties involved, such as the complex operator compositions, which require introducing new stream variables ("named windows" in EPL). The details of our compiler scheme are outside the scope of this paper, and will be described elsewhere.

V. Validation

This section presents the validation of our approach on the HomeAssist platform. The expressiveness of our DSL is validated by re-defining existing services in it. The correctness of our compiler is validated by comparing the results of executing the compiled rules with the results of existing services deployed in the platform. Finally, the efficiency of our DSL is validated by measuring some performance figures of our running implementation.

A. Expressiveness

To validate the expressiveness of our DSL, we re-implemented 53 services already deployed in HomeAssist. These services are variations of 13 families of rules, which are listed in Figure 4. Variations within each family were required to cover all the sensors in the home and their combinations. For example, variations of the rule "Long inactivity" are required for key rooms other than the bedroom, such as the living room or toilets. Similarly, the rule "Presence dependency" requires a presence in the kitchen to be detected when any appliance is used, such as the fridge, coffeemaker, or stove.

Rewriting an extended range of services allowed us to validate that our DSL and its underlying concepts (event, state, Allen's operators) are expressive enough to cover realistic services in the domain of aging in place.

B. Correctness

We did not attempt to prove the correctness of our DSL compiler with respect to a formal semantics of its operators, although this work would be of interest. Instead, we empirically validated the correctness of the compiled services by a combination of visual code inspection and extensive testing. We performed manual inspection of all the intermediate forms described earlier (core DSL, EPL pseudo-code, Esper EPL) to ensure that they remain consistent with their original counterparts.

Then, we validated the compiler output (*i.e.,* resulting EPL rules) in two phases. First, we tested the rules by executing them on log files from the HomeAssist project. These logs contain the timestamped events produced by all the sensors in the infrastructure, whether hardware or software. The results were checked automatically for correctness using Perl scripts, implementing the same service specifications. Note that these scripts are much simpler to write than the real applications, as they execute on log histories of sensed data, so they do not have to compute results online, and must not deal with the sensor infrastructure.

We repeatedly compared the results produced by the compiled DSL services and the scripted specifications on extended log histories. This iterative process allowed us to refine our compilation schemas until it produced the same results as the scripted specifications.

Name	Description / DSL	Metrics				Stakeholders
		DSL		EPL		
		# events	# states	# events	# not	
Presence dependency	Detect if cupboard status changes while no presence in kitchen `Cupboard becomes open occurs while Presence(Kitchen) is false`	1	1	2	1	Sensor installer
Departure alert	Detect if entrance door is opened at least for 5 minutes during calendar night time `Door is open for 5 minutes occurs while Night time`	1	1	2	2	Occupational therapist Caregiver
Door alert	Detect if entrance door is opened at least for 5 minutes during their is no presence in entrance `Door is open occurs for 5 min while Presence(Entrance) is false`	0	2	4	6	User Caregiver
Long inactivity	Detect if no movement in Bedroom since 24 hours `Presence(Bedroom) is false for 24 hours`	0	1	1	1	Occupational therapist Caregiver
Fridge opened	Detect if fridge remains open at least 5 minutes `Fridge is open for 5 minutes`	0	1	1	1	User Caregiver

Breakfast	Detect cupboard and coffeemaker opening (any order) during breakfast period
	`{Cupboard becomes open and CoffeeMaker becomes on}`
	`occurs while BreakfastTime`
		2	1	3	1	User Caregiver

Lunch reheat	Detect freezer opening and stove use in the 10 minutes following or freezer opening during stove use all during lunch period
	`{ ( Freezer becomes open precedes within 10 minutes`
	`Stove becomes on )`
	`or`
	`( Freezer becomes open occurs while`
	`Stove is on ) }`
	`occurs while Lunch Time`
		3	2	5	3	Caregiver User

Dinner	Detect fridge opening and microwave use (any order) during dinner period
	`{Fridge becomes open and Microwave becomes on}`
	`occurs while Dinner Time`
		2	1	2	1	User Caregiver

Go to bed	Detect end of presence in bathroom and begin of presence in bedroom in the 10 minutes following during go-to-bed period
	`( Presence(Bathroom) becomes false precedes within 10 minutes`
	`Presence(Bedroom) becomes true )`
	`occurs while Go-to-bed Time`
		2	1	3	2	Caregiver User

Wake-up	Detect end of presence in bedroom and begin of presence in kitchen in the 10 minutes following during go-to-bed period
	`( Presence(Bedroom) becomes false precedes within 10 minutes`
	`Presence(Kitchen) becomes true )`
	`occurs while Wake-up Time`
		2	1	3	2	Caregiver User

Commfailure warning	Detect any sensor that fails to communicate `Commfailure( Any ) becomes true`	1	0	1	0	Platform maintainer
Commfailure alert	Detect any sensor that has failed to communicate since 24 hours `Commfailure( Any ) is true for 24 hours`	0	1	1	1	Platform maintainer Sensor installer
Battery alert	Detect battery level of any senser that become less than 5% `BatteryLevel( Any ) becomes less than 5`	1	0	1	0	Sensor installer

Fig. 4. Services examples

In a second phase, we connected the compiled DSL services to the online stream of events in the production platform for 9 users during 1 month, in parallel with the existing services written in Java. No difference was observed between the results of both systems (Java and DSL).

C. Performance

To validate the applicability of our DSL approach in practice, we measured the performance of the EPL rules produced by our DSL compiler with respect to three indicators: response time to online events, processing time, and memory consumption. The rules were executed on a PC equipped with an Intel Core i5-3320MHz and 8Go of RAM, running the Linux kernel 4.15, and Esper 5.5. For easier generalization of the results, all the measurements were performed by forcing execution on a single core.

The response time of a rule indicates the time between the last event that should trigger a rule, and its effective triggering. A low response time means that the rule is sufficiently reactive for practical use. We measured this latency by executing the 53 rules on a log of 1 year of one home, respectively 10 homes multiplexed together. Time was accelerated by two different factors to simulate higher traffic, i.e., the recorded events in the log were submitted N times more quickly. As can be seen from the results in Table I, the maximum response time of all our rules were always less than a second. This order of magnitude is perfectly compatible with the kind of rules that are implemented in this platform for aging in place. Moreover, the average latency is between 3 and 12 milliseconds.

To ensure that our implementation can scale up to hundreds of users and to tens of implemented services, we measured CPU and memory consumption in batch mode for various logs of 1 year from homes of various sizes (H1 to H5), ranging from small apartments to houses with several floors. That is, all events in the logs were submitted sequentially with no delay, and the rules were slightly modified to replace timers with

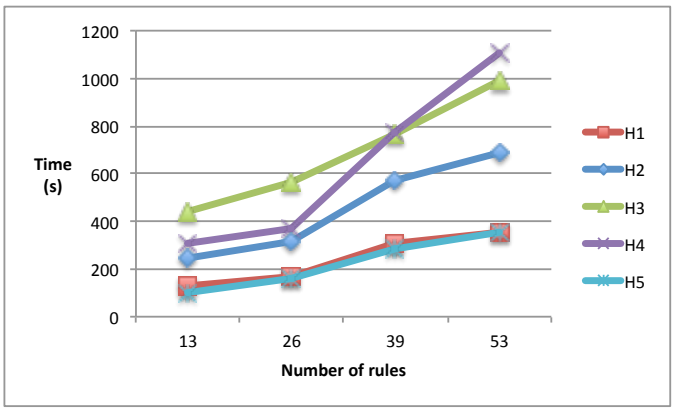

Fig. 5. Total CPU load (in seconds) for processing a 1 year log, when varying the number of rules

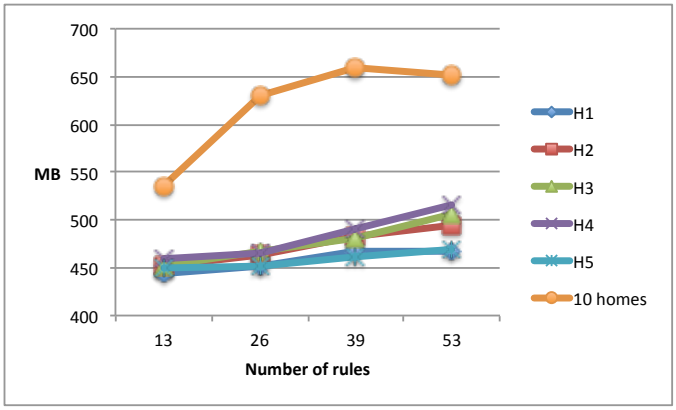

Fig. 6. Maximum memory consumption (in MBytes) for processing a 1 year log, when varying the number of rules

computations on event timestamps.

We first studied the variation of the batch processing time according to the number of rules. For that purpose, we created subsets of the full set of 53 rules containing 13, 26, and 39 rules. As can be seen from Figure 5, the total processing time for executing all the rules on 1 year spans from 5 to 20 minutes, depending on the home. Processing time increases with the number of rules according to a mostly linear pattern, except for one home (H5), where some rules (belonging to r2, r3, r4) require more processing than others (belonging to r1). As processing 1 year takes less than 20 minutes, a single CPU core could process hundreds of homes simultaneously.

We then measured the memory consumption of our implementation in the same batch conditions. The results in Figure 6 show that memory consumption for various single homes is at most half a GB. Moreover, memory consumption increases much less than linearly with the number of homes, because 10 homes (including H1 to H5 and 5 other homes) are processed simultaneously with less then 0.7 GB. These figures show that a single server with 10GB memory could process more than 100 homes simultaneously.

Home(s)	Acceleration factor	Max latency (ms)	Avg latency (ms)
H4	100,000	909	3.5
	1,000,000	900	3.8
10 homes	100,000	941	11.9
(H1…H10)	1,000,000	942	11.9

TABLE I
LATENCY FOR RULE TRIGGERING

VI. RELATED WORK

There are many works aiming to simplify and support the development of context-aware applications in smart homes. Let us classify them with respect to how they approach service development: user-oriented domain-specific languages, automata-oriented event processing approaches, and middleware-oriented approaches.

a) User-oriented, domain-specific approaches: These approaches start from the needs of end users of smart homes, and provide a domain-specific language for developing context-aware applications, usually complemented with a dedicated development environment. This end-user programming approach has resulted in both textual and visual languages. Scratch [16] offers visual programming notations, covering most of a general-purpose programming language. In principle, this language would allow users to write a wide range of programs in the domain of smart homes, but would not support such development with domain-specific abstractions. IFTTT (If This Then That)[3] offers much more specialized graphic notations for automating simple processes involving web services, sensors, and actuators. This approach lowers the end-user conceptual effort at the expense of drastically reducing the expressiveness of the language. For example, conditionals only consists of one sensor or service, and cannot be composed. Furthermore, no distinction is done between events and states, which has been shown to confuse users when defining services [17].

Improving on IFTTT, AppsGate [18] introduces a textual DSL, which makes a clear distinction between states and events, and allows some limited compositions of tests in rules. AppsGate provides an end-user development environment dedicated to smart homes and it has been shown to cover simple rules for comfort-oriented automation. While this extension of IFTTT is very promising, it still falls short of addressing the scope of real AAL applications. For example, the authors mention that "expressing compound conditions [involving several events or states] was difficult" [18, p.13]. Moreover, even simple temporal composition such as "A immediately precedes B" cannot be expressed in their language, due to the absence of boolean connectors in conditions (e.g., once A is found, wait for B but no A). Because our DSL approach is based on a domain analysis, we identified the common scenarios to be addressed and provide abstractions expressive enough to capture them.

Conceptually, an interesting case of top-down approach to context awareness is the identification of 6 top-level *context*

[3]https://ifttt.com

dimensions having the widest used in pervasive computing [1]: the physical world, the cyber-world, the user, his/her activities, the social context, and their dynamics. These context dimensions result from a commonality analysis performed on 13 context meta-models formalized in the literature, and about 300 research papers on pervasive computing applications. Our approach also involved a commonality analysis but in a specific area: smart home services for aging in place. Thanks to this domain instantiation, not only were we able to elicit common concepts, but we also designed a language and tool support that revolve around these concepts.

b) Automata-oriented, event processing approaches: Some approaches leverage existing models of automata and associated tools. Indeed, contexts may be modeled as particular sequences of events in the environment, constrained in their order and time delays. For instance, the situation of the unattended door may be recognized by an automaton accepting a door opening event and a door closing event, separated by a time delay greater than a given value. Such executable models of context recognition may be expressed in a DSL for automata modeling, which can be visual (StateCharts, SyncCharts) or textual (synchronous languages) [19]. Time delays must be handled explicitly in these models by using external timers to generate timeout events. Timed automata [20] add a native expression of time delays in the model. These automata-based models are usually accompanied by formal tools for proving useful temporal properties about the model, such as state reachability. They are expressive enough for handling all cases needed by smart home services, but require users to implement common patterns such as sequencing events, recognizing a set of events during a state, *etc.* When implementing these base patterns as timed automata, users may introduce subtle bugs or slight variations in behavior. This issue is of course amplified when the models are written by stakeholders with different levels of expertise.

Complex event processing languages [10] introduce event composition operators that implement some very common event patterns, such as ordered sequence of events within a time delay, and event alternatives. Some CEPs use interval-based semantics for complex events, and sometimes even define the Allen temporal relations between interval-based events [21]–[24]. However, there is no native notion of state in these CEPs. For instance, the state of a door being open has to be coded, as a complex event starting with a door opening and ending with a door closing, with no door action in between. This kind of encoding is error-prone and tends to yield intricate CEP formulas (as can be seen in our examples). This encoding is exactly what our DSL compiler automatically generates, in a uniform and predictable way, thereby providing a complete set of reliable common state/event operators. In fact, this compilation approach sets apart our work from most works in CEP, typically introducing a new CEP language with its standalone implementation, rather than translating in an existing, standard CEP language.

c) Middleware-oriented approaches: Some approaches to developing context-aware applications rely on a middleware (or a framework) to provide programmers, in a general programming language, with dedicated abstractions for operating and managing devices (*e.g.,* sensor discovery) and context-aware services; examples include FedNet [25], HomeOS [26], Gaia [27], Olympus [28], and Plan B [29].

Raising further the abstraction level, other approaches introduce a disciplined, declaration-based development process, dedicated to context-aware services. These approaches go as far as automating part of the programming task by generating a programming framework. Examples of tool-based development approaches are DiaSuite [14], [15] and IoTSuite [30].

All these approaches aim to abstract over sensor infrastructure details, but do not simplify the programming of context detection logic. The core of context processing revolves around recognizing specific patterns of events and states. Without specific support, this programming is low level, tedious and error-prone, even for seasoned programmers. Our approach provides the programmer with this specific support.

VII. Discussion

Our DSL bridges the gap between high-level domain concepts and low-level mechanisms of event handling. As a consequence, it contributes to making rules more concise and to simplify their development, by encapsulating details of event handling in a compiler. Indeed, the original Java applications implementing HomeAssist services contain manually implemented timed automata, which recognize the sequences of events corresponding to each DSL rule. Timing constraints are explicitly handled by using a timer service, producing timeout events that are inserted in the stream of events, produced by the sensor infrastructure. In our DSL rules, these low-level details of state and time handling are included in the semantics of operators. For instance, the role of explicit timers corresponds to our time-constrained operator variants. This lowers the efforts to write rules and to make them more predictable.

Limitations: Our approach is a first step towards simplifying the development of context-aware applications in the domain of aging in place, and presents a number of limitations.

First of all, our DSL only is dedicated to recognizing contexts. It provides no constructs for performing actions on the environment. These must be currently programmed in a generic programming language. It would be useful to extend our domain analysis to also cover the control part of typical applications, and to derive domain-specific concepts and notations to perform actions.

As far as applications are concerned, we have designed and tested our DSL only on services in the domain of aging in place, which involves a specific set of composition operators. However, our DSL can express rich, arbitrarily nested combinations. It would be interesting to apply it to other domains of context-aware applications in the future.

Finally, our rules always return boolean values. However, context-aware information may sometimes be more general than strictly binary. For instance, a daily activity such as meal preparation might be detected in a more nuanced way as a

probability between 0 and 1, to cope with some amount of deviations from the user's routine. Currently, in our DSL the different routine variations must be coded as different rules, which is not always practical. In the future, it would be interesting to consider extending our approach with operators returning non-boolean values.

VIII. Conclusion

We have presented a new approach for developing context-aware services in a smart home, by analyzing a range of existing data processing layers in the domain of aging in place. We have identified key concepts and operations specific to context-aware processing. Based on this analysis, we have introduced a context-aware, domain-specific language and its software architecture, which allow to put in synergy the stakeholders of a context-aware home by providing them with a unified approach to designing and developing services. Our approach offers context aware-specific abstractions and notations, within a data-centric and data-driven paradigm.

We have validated our approach by applying it to an assisted living platform for aging in place. In particular, we have used our domain-specific language to re-implement existing services of the assisted living platform. These services were deployed and successfully tested for their effectiveness in performing the specific tasks of the stakeholders, such as: detection of daily activities, user risks, and sensor failures.

References

[1] C. Bauer, "A comparison and validation of 13 context meta-models." in *ECIS*, 2012, p. 17.

[2] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan, "Casas: A smart home in a box," *Computer*, vol. 46, no. 7, pp. 62–69, 2013.

[3] J. Feminella, D. Pisharoty, and K. Whitehouse, "Piloteur: a lightweight platform for pilot studies of smart homes," in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. ACM, 2014, pp. 110–119.

[4] A. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home automation in the wild: challenges and opportunities," in *proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 2115–2124.

[5] J. Coutaz, E. Fontaine, N. Mandran, and A. Demeure, "Disqo: A user needs analysis method for smart home," in *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*. ACM, 2010, pp. 615–618.

[6] P. Rashidi and A. Mihailidis, "A survey on ambient-assisted living tools for older adults," *IEEE journal of biomedical and health informatics*, vol. 17, no. 3, pp. 579–590, 2013.

[7] L. Caroux, C. Consel, L. Dupuy, and H. Sauzéon, "Verification of Daily Activities of Older Adults: A Simple, Non-Intrusive, Low-Cost Approach," in *ASSETS - The 16th International ACM SIGACCESS Conference on Computers and Accessibility*, Rochester, NY, United States, Oct. 2014, pp. 43–50.

[8] C. Consel, L. Dupuy, and H. Sauzéon, "HomeAssist: An assisted living platform for aging in place based on an interdisciplinary approach," in *Proceedings of the 8th International Conference on Applied Human Factors and Ergonomics (AHFE 2017)*. Springer, 2017.

[9] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, Nov. 1983.

[10] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 15:1–15:62, Jun. 2012.

[11] J. Kaye, "Making pervasive computing technology pervasive for health & wellness in aging," *Public Policy & Aging Report*, 2017.

[12] J. A. Kaye, S. A. Maxwell, N. Mattek, T. L. Hayes, H. Dodge, M. Pavel, H. B. Jimison, K. Wild, L. Boise, and T. A. Zitzelberger, "Intelligent systems for assessing aging changes: home-based, unobtrusive, and continuous assessment of aging," *Journals of Gerontology Series B: Psychological Sciences and Social Sciences*, vol. 66, no. suppl_1, pp. i180–i190, 2011.

[13] C. Consel, L. Dupuy, and H. Sauzéon, "A unifying notification system to scale up assistive services," in *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*. ACM, 2015, pp. 77–87.

[14] B. Bertran, J. Bruneau, D. Cassou, N. Loriant, E. Balland, and C. Consel, "DiaSuite: A tool suite to develop Sense/Compute/Control applications," *Science of Computer Programming*, vol. 79, pp. 39–51, 2014.

[15] D. Cassou, J. Bruneau, C. Consel, and E. Balland, "Toward a tool-based development methodology for pervasive computing applications," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1445–1463, 2012.

[16] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, "Scratch: Programming for all," *Commun. ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.

[17] J. Huang and M. Cakmak, "Supporting mental model accuracy in trigger-action programming," in *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, ser. UbiComp '15. New York, NY, USA: ACM, 2015, pp. 215–225.

[18] J. Coutaz and J. L. Crowley, "A first-person experience with end-user development for smart homes," *IEEE Pervasive Computing*, vol. 15, no. 2, pp. 26–39, 2016.

[19] A. Gamatié, *Synchronous Programming: Overview*. New York, NY: Springer New York, 2010, pp. 21–39.

[20] J. Bengtsson and W. Yi, *Timed Automata: Semantics, Algorithms and Tools*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 87–124.

[21] D. Anicic, P. Fodor, S. Rudolph, R. Stühmer, N. Stojanovic, and R. Studer, *A Rule-Based Language for Complex Event Processing and Reasoning*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 42–57.

[22] M. Li, M. Mani, E. A. Rundensteiner, and T. Lin, "Complex event pattern detection over streams with interval-based temporal semantics," in *Proceedings of the Fifth ACM International Conference on Distributed Event-Based Systems, DEBS 2011, New York, NY, USA, July 11-15, 2011*, D. M. Eyers, O. Etzion, A. Gal, S. B. Zdonik, and P. Vincent, Eds. ACM, 2011, pp. 291–302.

[23] S. Helmer and F. Persia, "High-level surveillance event detection using an interval-based query language," in *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, Feb 2016, pp. 39–46.

[24] S. Hausmann, "The language dura: A declarative event query language for reactive event processing," 2014.

[25] F. Kawsar, T. Nakajima, and K. Fujinami, *Deploy spontaneously: Supporting end-users in building and enhancing a smart home*. New York, NY, USA: ACM, 2008, pp. 282–291.

[26] N. Rosen, R. Sattar, R. W. Lindeman, R. Simha, and B. Narahari, "Homeos: Context-aware home connectivity," in *Proceedings of the International Conference on Wireless Networks, ICWN '04, Volume 2 & Proceedings of the International Conference on Pervasive Computing and Communications, PCC'04, June 21-24, 2004, Las Vegas, Nevada, USA*, H. R. Arabnia, L. T. Yang, and C. Yeh, Eds. CSREA Press, 2004, pp. 739–744.

[27] M. Rom'an, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, "A middleware infrastructure for active spaces," *Pervasive ComputingPervasive Computing*, vol. Middleware, no. October-November, 2002.

[28] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R. H. Campbell, and M. D. Mickunas, "Olympus: A high-level programming model for pervasive computing environments," in *Third IEEE International Conference on Pervasive Computing and Communications*, March 2005, pp. 7–16.

[29] F. J. Ballesteros, E. Soriano, K. L. Algara, and G. G. Muzquiz, "Plan B: an operating system for ubiquitous computing environments," in *4th IEEE International Conference on Pervasive Computing and Communications (PerCom 2006), 13-17 March 2006, Pisa, Italy*. IEEE Computer Society, 2006, pp. 126–135.

[30] P. Patel and D. Cassou, "Enabling high-level application development for the internet of things," *Journal of Systems and Software*, vol. 103, pp. 62 – 84, 2015.